EXHIBIT A

```
; Last Modified Feb 11, 1999.
; This is the Code for the new Tubular
; Motor logic board using a triac.

;*********************************************************************
****
;
;     Equate Statements
;
;*********************************************************************
****

      globals on

P01M_INIT    .equ  00000100B
P2M_INIT_    .equ  01100011B
P3M_INIT     .equ  00000001B
P01S_INIT    .equ  00001010B
P2S_INIT     .equ  00000000B
P3S_INIT     .equ  00000000B
P2M_EEOUT    .equ  11111011B   ; Mask for outputting data to EEPROM
P2M_EEIN     .equ  00000100B   ; Same for input

;--------------------------------------------------------------------
----
; GLOBAL REGISTERS
;--------------------------------------------------------------------
----
;*********************************************************************
****
; LEARN EE GROUP REGISTERS FOR LOOPS ECT
;*********************************************************************
****

LEARNEE_GRP .equ   20H
P2M_SHADOW   .equ  LEARNEE_GRP+0      ; Mask for mode of P2
TEMP         .equ  LEARNEE_GRP+2      ;
MTEMPH       .equ  LEARNEE_GRP+6      ; memory temp
MTEMPL       .equ  LEARNEE_GRP+7      ; memory temp
MTEMP        .equ  LEARNEE_GRP+8      ; memory temp
SERIAL       .equ  LEARNEE_GRP+9      ; serial data to and from nonvol
memory
ADDRESS      .equ  LEARNEE_GRP+10     ; address for the serial nonvol
memory

temp         .equ  r2     ;
mtemph       .equ  r6     ; memory temp
mtempl       .equ  r7     ; memory temp
mtemp        .equ  r8     ; memory temp
serial       .equ  r9     ; serial data to and from nonvol memory
address      .equ  r10    ; address for the serial nonvol memory

;*****************************************************************
MAIN_GRP     .equ  30H

UP_LIMIT_H         .equ  MAIN_GRP+0   ; upper limit high byte
UP_LIMIT_L         .equ  MAIN_GRP+1   ; upper limit low byte
UP_LIMIT           .equ  MAIN_GRP+0   ; upper limit word
```

```
DOWN_LIMIT_H        .equ   MAIN_GRP+2   ; lower limit high byte
DOWN_LIMIT_L        .equ   MAIN_GRP+3   ; lower limit low byte
DOWN_LIMIT          .equ   MAIN_GRP+2   ; lower limit word
POS_CNTR_H          .equ   MAIN_GRP+4   ; position counter high byte
POS_CNTR_L          .equ   MAIN_GRP+5   ; position counter low byte
POS_CNTR            .equ   MAIN_GRP+4   ; position counter
PP_DIST             .equ   MAIN_GRP+6   ; is 180.
HALF_PP_DIST        .equ   MAIN_GRP+7   ; 80
POS_CNTR_TEMP_H     .equ   MAIN_GRP+8   ; temp counter for FIRST_TIME
POS_CNTR_TEMP_L     .equ   MAIN_GRP+9   ; temp counter for FIRST_TIME
POS_CNTR_TEMP       .equ   MAIN_GRP+8   ; temp counter for FIRST_TIME
UP_AND_DOWN         .equ   MAIN_GRR+10  ; (A) tells us if up or down or
                                        ;     both buttons are pushed.
RESET_FLAG          .equ   MAIN_GRP+11  ; (B) tells us if reset is pushed
UP_DEBOUNCER        .equ   MAIN_GRP+12  ; (C) up debouncer
DOWN_DEBOUNCER      .equ   MAIN_GRP+13  ; (D) down debouncer
POWER_DEBOUNCER     .equ   MAIN_GRP+14  ; (E) power debouncer
TAP_CNTR            .equ   MAIN_GRP+15  ; (F) tap counter
AllIntOn            .equ   MAIN_GRP+16  ; (0) sets up interrupts
OFF_LFC             .equ   MAIN_GRP+17  ; (1) off power line sampler.
LF_TIMER            .equ   MAIN_GRP+18  ; (2) Line Filter Timer
UP_LFC              .equ   MAIN_GRP+19  ; (3) up direction sampler
DOWN_LFC            .equ   MAIN_GRP+20  ; (4) down direction sampler
POWER_LFC           .equ   MAIN_GRP+21  ; (5) power line sampler.
MOTOR_FLAG          .equ   MAIN_GRP+22  ; (6) Used for a counter/timer.
LEARNED             .equ   MAIN_GRP+23  ; (7) Tells us if first time
PPOINT              .equ   MAIN_GRP+24  ; (8) high if pass point seen
RPM_DEBOUNCER_H     .equ   MAIN_GRP+25  ; (9) RPM signal high debouncer.
IR_TIMER            .equ   MAIN_GRP+26  ; (A) timer for triac enable delay
STOP_FLAG           .equ   MAIN_GRP+27  ; (B) tells main loop to stop
START_FLAG          .equ   MAIN_GRP+28  ; (C) flag to start power sampling
PP_DEBOUNCER_H      .equ   MAIN_GRP+29  ; (D) pass point signal high
debouncer.
PP_DEBOUNCER_L      .equ   MAIN_GRP+30  ; (E) pass point signal low
debouncer.
RPM_DEBOUNCER_L     .equ   MAIN_GRP+31  ; (F) RPM signal low debouncer.
UP_LIMIT_FLAG       .equ   MAIN_GRP+32  ; (0) hit up limit flag.
DOWN_LIMIT_FLAG     .equ   MAIN_GRP+33  ; (1) hit down limit flag.
STALL_FLAG          .equ   MAIN_GRP+34  ; (2) no pulses for 2 sec.
RPM_PULSE           .equ   MAIN_GRP+35  ; (2) =100ms pulse after travel.

;************************************************************************
CHECK_GRP   .equ   70H

check_sum_value    .equ   018H

check_sum    .equ  r0
rom_data     .equ  r1
test_adr_hi  .equ  r2
test_adr_lo  .equ  r3
test_adr     .equ  rr2


STACKEND     .equ  0A0H   ; start of the stack
STACKTOP     .equ  238    ; end of the stack


csh          .equ  00010000B ; chip select high for the 93c46
csl          .equ  11101111B ; chip select low for 93c46
clockh       .equ  00001000B ; clock high for 93c46
clockl       .equ  11110111B ; cl ck low for 93c46
```

```
doh           .equ  00000100B ; data out high for 93c46
dol           .equ  11111011B ; data out low for 93c46
psmask        .equ  01000000B ; mask for the program switch
csport        .equ  P2      ; chip select port
dioport       .equ  P2      ; data i/o port
clkport       .equ  P2      ; clock port
psport        .equ  P2      ; program switch port
MemoryTimer   .equ  96      ;

WATCHDOG_GROUP    .equ   0FH
PCON          .equ  r0
SMR           .equ  r11
WDTMR         .equ  r15

FILL  .macro
      .byte 0FFh
      .endm
FILL10       .macro
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .endm
FILL50       .macro
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
      .byte 0FFh
       byte 0FFh
```

```
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .byte 0FFh
        .endm

TRAP    .macro
        jp     start
        jp     start
        jp     start
        jp     start
        jp     start
        .endm

TRAP10       .macro
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        TRAP
        .endm

;*************************************************************************
*
;*
;*                         Interrupt Vector
Table
;*
;*************************************************************************
*


        .org  0000H

        .word TIMER1_INT               ;IRQ0, P3.2
        .word TIMER1_INT               ;IRQ1, P3.3
        .word TIMER1_INT               ;IRQ2, P3.1
        .word TIMER1_INT               ;IRQ3, P3.0
        .word TIMER0_INT               ;IRQ4, T0
        .word TIMER1_INT               ;IRQ5, T1
```

```
      .page

            . rg    000CH

;*************************************************************************
*
; REGISTER INITILIZATION
;*************************************************************************
*

start:
START:

      di                         ; turn off the interrupt for init
      ld     RP,#WATCHDOG_GROUP
      ld     WDTMR,#00000111B    ; rc dog 100mS
      clr    RP                  ; clear the register pointer
      WDT                        ; kick the dog

      xor    P2, #10000000B      ; toggle pin 3.

;*************************************************************************
*
;* PORT INITIALIZATION
;*************************************************************************
*

      ld     P01M,#P01M_INIT           ; set mode p00-p03 out p04-p07in
      ld     P3M,#P3M_INIT             ; set port3 p30-p33 input analog
mode
                                       ; p34-p37 outputs
      ld     P2M,#P2M_INIT             ; set port 2 mode
      ld     P2M_SHADOW, #P2M_INIT     ; Set readable register
      ld     P0,#P01S_INIT             ; RESET all ports
      ld     P2,#P2S_INIT              ;
      ld     P3,#P3S_INIT              ;

;*************************************************************************
*
;*                 Internal RAM Test and Reset All RAM =    mS
*
;*************************************************************************
*

      jp     STACK
      srp      #0F0H           ; POINT to control register group
      ld       r15, #4         ; r15= pointer (minimum of RAM)

write_again:

      WDT                      ; kick the dog
      xor    P2, #10000000B    ; toggle pin 3.
      ld       r14,#1

write_again1:

      ld       @r15,r14        ; write 1,2,4,8,10,20,40,80
      cp       r14,@r15        ; then compare
```

```
        jp      ne,system_error
        rl      r14
        jp      nc,write_again1
        clr     @r15                    ; write RAM(r5)=0 to memory
        inc     r15
        cp      r15,#240
        jp      ult,write_again

;*****************************************************************
*
;*           Checksum Test                                      *
;*****************************************************************
*

CHECKSUMTEST:

        srp     #CHECK_GRP
        ld      test_adr_hi,#0FH
        ld      test_adr_lo,#0FFH ; maximum address=fffh

add_sum:

        WDT                         ; kick the dog
        xor     P2, #10000000B      ; toggle pin 3.
        ldc       rom_data,@test_adr     ;read ROM code one by one
        add       check_sum,rom_data     ;add it to checksum register
        decw      test_adr               ;increment ROM address
        jp        nz,add_sum             ; address=0 ?
        cp        check_sum,#check_sum_value
        jp        z,system_ok             ;check final checksum = 00 ?

system_error:

        and     P0,#11111011B           ; turn on the led
        jp      system_error

        .byte        256-check_sum_value

system_ok:

        WDT                         ; kick the dog
        xor     P2, #10000000B      ; toggle pin 3.

        ld      STACKEND,#STACKTOP      ; start at the top of the stack

SETSTACKLOOP:

        ld      @STACKEND,#01H          ; set the value for the stack vector
        dec     STACKEND                ; next address
        cp      STACKEND,#STACKEND      ; test for the last address
        jp      nz,SETSTACKLOOP         ; loop till done

;*****************************************************************
*
;*STACK INITILIZATION
;*****************************************************************
*

STACK:
      WDT                         ; kick the dog
      xor   P2, #10000000B        ; toggle pin 3.
      clr   254
      ld    255,#238              ; set the start of the stack

;********************************************************************
*
;*TIMER INITILIZATION
;********************************************************************
*

TIMER:

       ld    PRE0,#00010001B    ; set the prescaler to 1/4 for 250Khz
       ld    T0, #0FAH          ; set the counter to count 250 to 0
(T0=1ms)
       ld    PRE1,#11111100B    ; set the prescaler to 1/63 for 16Khz
       ld    T1, #0FFH          ; set the counter to count 256 to 0
(T1=16ms)
       ld    TMR,#00000111B     ; load timers with initial values.

;********************************************************************
*
;*PORT INITIALIZATION
;********************************************************************
*

       ld    P01M,#P01M_INIT          ; set mode p00-p03 out p04-p07in
       ld    P3M,#P3M_INIT            ; set port3 p30-p33 input analog
mode
                                      ; p34-p37 outputs
       ld    P2M,#P2M_INIT            ; set port 2 mode
       ld    P2M_SHADOW, #P2M_INIT    ; Set readable register
       ld    P0,#P01S_INIT            ; RESET all ports
       ld    P2,#P2S_INIT             ;
       ld    P3,#P3S_INIT             ;

;********************************************************************
*
; INITERRUPT INITILIZATION
;********************************************************************
*

SETINTERRUPTS:

       ld    IPR,#00101101B           ; set the priority to RPM
       ld    IMR,#01010000B           ; set IMR for T0 interrupt only
       ld    IRQ,#01000000B           ; set the edge, clear int


;********************************************************************
*
; SET SMR & PCON
;********************************************************************
*

       ld    RP, #WATCHDOG_GROUP
       ld    SMR,#00011110B           ; recovery s urce = P2 NOR 0:7
       ld    PCON,#10010110B          ; reset the pcon no comparator
output
```

```
                                    ; STANDARD emi mode
        clr     RP

;**********************************************************************
*
; VARIALBE INITILIZATION
;**********************************************************************
*

        ld      UP_LIMIT_H, #01
        ld      UP_LIMIT_L, #00
        ld      DOWN_LIMIT_H, #255
        ld      DOWN_LIMIT_L, #00
        ld      POS_CNTR_H, #00
        ld      POS_CNTR_L, #00
        ld      POS_CNTR_TEMP_H, #00
        ld      POS_CNTR_TEMP_L, #00
        ld      LF_TIMER, #00
        ld      OFF_LFC, #00
        ld      UP_LFC, #00
        ld      DOWN_LFC, #00
        ld      POWER_LFC, #00
        ld      MOTOR_FLAG,#00
        ld      LEARNED, #02
        ld      PPOINT, #00
        ld      IR_TIMER, #00
        ld      STOP_FLAG, #00
        ld      START_FLAG, #00
        ld      UP_AND_DOWN, #00
        ld      RESET_FLAG, #00
        ld      UP_DEBOUNCER, #00
        ld      DOWN_DEBOUNCER, #00
        ld      POWER_DEBOUNCER, #00
        ld      TAP_CNTR, #00
        ld      UP_LIMIT_FLAG, #00
        ld      DOWN_LIMIT_FLAG, #00
        ld      PP_DEBOUNCER_L, #00
        ld      RPM_DEBOUNCER_L, #00
        ld      STALL_FLAG, #00
        ld      RPM_PULSE, #00

        ld      TEMP, #00
        ld      MTEMPH, #00
        ld      MTEMPL, #00
        ld      MTEMP, #00
        ld      SERIAL, #00
        ld      ADDRESS, #00

        ld      PP_DIST, #180
        ld      HALF_PP_DIST, #79
        ld      AllIntOn, #01010000B      ; just enable timer at first
        ld      PP_DEBOUNCER_H, #31
        ld      RPM_DEBOUNCER_H, #11


;**********************************************************************

; READ THE MEMORY 2X
;**********************************************************************
```

```
        ei

WAIT_BEFORE_READING:

        cp      LF_TIMER, #20
        jp      ne, WAIT_BEFORE_READING

        WDT                             ; kick the dog
        xor     P2, #10000000B          ; toggle pin 3.

        and     TMR,#11111101B          ; disable timer 0
        di
        ld      ADDRESS, #00            ; this address contains UP_LIMIT
        nop
        call    READMEMORY              ; read the value 2X 1X INIT
        nop
        call    READMEMORY              ; read the value
        ld      UP_LIMIT_H, MTEMPH
        ld      UP_LIMIT_L, MTEMPL
        ld      ADDRESS, #01            ; this address contains DOWN_LIMIT
        nop
        call    READMEMORY              ; read the value
        ld      DOWN_LIMIT_H, MTEMPH
        ld      DOWN_LIMIT_L, MTEMPL
        ld      ADDRESS, #02            ; this address contains POS_CNTR
        nop
        call    READMEMORY              ; read the value
        ld      POS_CNTR_H, MTEMPH
        ld      POS_CNTR_L, MTEMPL
        nop
        WDT                             ; kick the dog
        xor     P2, #10000000B          ; toggle pin 3.
        nop
        ld      ADDRESS, #04            ; this address contains LEARNED
        nop
        call    READMEMORY              ; read the value
        ld      RESET_FLAG, MTEMPH
        ld      LEARNED, MTEMPL
        ld      ADDRESS, #05            ; this address contains PPOINT
        nop
        call    READMEMORY              ; read the value
        ld      PPOINT, MTEMPH
        nop
        ld      ADDRESS, #06            ; this address contains the LIMIT_FLAG s
        nop
        call    READMEMORY              ; read the value
        ld      UP_LIMIT_FLAG, MTEMPH
        ld      DOWN_LIMIT_FLAG, MTEMPL
        nop
        ld      ADDRESS, #03                    ; this address contains TAP_CNTR
        nop
        call    READMEMORY                      ; read the value
        ld      TAP_CNTR, MTEMPH
        clr     MTEMPL

        WDT                             ; kick the dog
        xor     P2, #10000000B          ; toggle pin 3.
        or      TMR,#00000010B          ; enable timer 0

;********************************************************************************
```

```
;*
;*   CHECK IF ANY MODE FLAGS ARE SET.   IF SO, JUMP TO THAT MODE.
;*   ELSE, CHECK IF TAP_COUNTER HAS REACHED 5 TAPS.  IF SO, SET LEARN
MODE
;*   FLAG.
;*****************************************************************
;*

        WDT                            ; kick the dog
        xor    P2, #10000000B             ; toggle pin 3.
        cp     RESET_FLAG, #85            ; see if in reset mode
        jp     eq, PASSPOINT_RESET        ; if so, goto passpoint reset
        cp     LEARNED, #02               ; see if learn mode flag set
        jp     ult, CHECK_FOR_TAP_HIGH    ; if so, go check tap count value

        cp     TAP_CNTR, #05              ; see if erase was pushed
        jp     eq, NOTE_ERASE             ; if so, goto clear limits
        ld     START_FLAG, #01            ; set flag for timer
        jp     CLEAR_UP_AND_DOWN

NOTE_ERASE:

; set LEARNED byte to 01 for learn mode
; WRITE TO MEMORY -- write LEARNED=1 to E^2

        ld     ADDRESS, #04               ; POINT TO ADDRESS THAT CONTAINS
LEARNED
        ld     MTEMPH, #00                ; load temp register with RESET_FLAG
byte
        ld     MTEMPL, #01                ; load temp register with LEARNED
byte
        nop
        call   WRITEMEMORY                ;
        ld     LEARNED, #01               ; set LEARNED to 1.
        jp     FIRST_TIME

NOTE_RESET:

; WRITE RESET_FLAG = 85 TO E^2.

        WDT                            ; kick the dog
        xor    P2, #10000000B             ; toggle pin 3.
        ld     ADDRESS, #04               ; POINT TO ADDRESS THAT CONTAINS
LEARNED
        ld     MTEMPH, #85                ; load temp register with RESET_FLAG
byte
        ld     MTEMPL, #00                ; load temp register with LEARNED
byte
        nop
        call   WRITEMEMORY                ;
        jp     PASSPOINT_RESET

CHECK_FOR_TAP_HIGH:

        cp     TAP_CNTR, #09              ; see if reset mode requested
        jp     eq, NOTE_RESET             ; if so, goto clear limits
        jp     FIRST_TIME
```

```
CLEAR_UP_AND_DOWN:

;*******************************************************************
*
;                               MAIN
LOOP
; THIS PORTION OF THE CODE JUST EXECUTES NORMAL OPERATION OF THE LOGIC
BOARD.
; NORMAL OPERATION IS TURNING ON THE TRIAC UNTIL EITHER THE UP OR DOWN
LIMIT IS
; REACHED OR POWER HAS BEEN RELEASED.
;
;*******************************************************************
*



PASSPOINT_RESET:

;****************************************************************
; THIS PORTION OF THE CODE RESETS THE PASS POINT GEARS TO THERE INITIAL
; SETTING.  IN ORDER TO BE IN THIS ROUTINE,THE POWER BUTTON MUST HAVE .
; BEEN PRESSED FOR LESS THAN 500 ms AT LEAST 9 CONSECUTIVE TIMES.
; AS A RESULT, THE RESET FLAG IS SET.
; AT THE CONCLUSION OF THIS ROUTINE, THE FLAG IS ERASED.
;****************************************************************



FIRST_TIME:

;****************************************************************
; THIS PORTION OF THE CODE LEARNS THE LIMIT OPPOSITE OF THE DIRECTION
; OF TRAVEL. IN ORDER TO BE IN THIS ROUTINE,THE POWER BUTTON MUST HAVE
; BEEN PRESSED FOR LESS THAN 500 ms BETWEEN 5-8 CONSECUTIVE TIMES.
; AS A RESULT, THE LEARNED FLAG IS SET.
; AT THE CONCLUSION OF THIS ROUTINE, THE FLAG IS ERASED.
;
;****************************************************************

;-------------------------------------------------------------------
------
;           THIS IS THE TIMER0 (HEARTBEAT) INTERRUPT ROUTINE
;           THIS ROUTINE IS ENTERED EVERY 1ms.
;-------------------------------------------------------------------
------

TIMER0_INT:

     ld    IMR, AllIntOn          ; turn on all the interrupts

CHECK_START_FLAG:
        inc     DELAY_TIMER                 ; increment line filter timer.
        cp      START_FLAG, #01             ; ready to check inputs?
        jp      n , TIMER0_RETURN           ; if not, leave.
        tm      P2, #00100000B              ; is POWER (P25) high?
        jp      z, INC_OFF_LFC              ; if not, don't sample up/dn pins.
        inc     POWER_LFC                   ; else, increment TOTAL_LFC.
        clr     OFF_LFC

TEST_MOTOR:

        cp      MOTOR_FLAG, #0AAH           ; is motor on?
        jp      eq, TIMER0_RETURN           ; if so, jump.
        tm      P2, #00000010B              ; is up (P21) input high?
        jp      z, TEST_DOWN_LFC            ; if not, don't inc UP_LFC.
        inc     UP_LFC                      ; else, increment DOWN_LFC.
        jp      TEST_POWER_LFC

TEST_DOWN_LFC:

        tm      P2, #00000001B              ; is down (P20)input high?
        jp      z, TEST_POWER_LFC           ; if not, don't inc UP_LFC.
        inc     DOWN_LFC                    ; increment DOWN_LFC
        jp      TEST_POWER_LFC

INC_OFF_LFC:

        inc     OFF_LFC                     ; increment OFF COUNTER
        clr     UP_LFC                      ; clear up counter
        clr     DOWN_LFC                    ; clear down counter
        clr     POWER_LFC                   ; clear power counter
        cp      OFF_LFC, #41                ; is counter at 41ms?
        jp      ne, TIMER0_RETURN           ; if so, then jump.
        jp      CHECK_FOR_POWER

TEST_POWER_LFC:

        cp      POWER_LFC, #04              ; is POWER_LFC more than 04?
        jp      ne, TIMER0_RETURN           ; if so, leave interrupt
        clr     OFF_LFC
        cp      POWER_DEBOUNCER, #22        ; is DB already at 22?
        jp      eq, CHECK_UP_LFC            ; if so, don't increment
        inc     POWER_DEBOUNCER             ; else, increment POWER DB
        cp      POWER_DEBOUNCER, #03        ; is UP DB at 3?
        jp      ne, CHECK_UP_LFC            ; if not, jump.
        inc     TAP_CNTR                    ; else, increment TAP_COUNTER
        jp      CHECK_UP_LFC                ; and jump.

CHECK_UP_LFC:

        cp      UP_LFC, #04                 ; is UP LFC at 3?
        jp      ult, CHECK_DOWN_LFC         ; if not, jump.
        cp      UP_DEBOUNCER, #255          ; is UP DB maxed out.
        jp      eq, SET_UP_AND_DOWN_FLAG    ; if so, jump.
        clr     DOWN_DEBOUNCER              ; clear debouncers
        inc     UP_DEBOUNCER                ; increment db
        cp      UP_DEBOUNCER, #22           ; if at 22, then set high.
        jp      ne, SET_UP_AND_DOWN_FLAG    ; else, skip.
        ld      UP_DEBOUNCER, #255          ; ld DB with 255.
        clr     TAP_CNTR                    ; clear TAP_COUNTER
        jp      SET_UP_AND_DOWN_FLAG
```

```
CHECK_DOWN_LFC:

        cp    DOWN_LFC, #04            ; is DOWN_LFC at 3?
        jp    ult, SET_UP_AND_DOWN_FLAG ; if not, jump.
        cp    DOWN_DEBOUNCER, #255     ; is DOWN DB maxed out.
        jp    eq, SET_UP_AND_DOWN_FLAG ; if so, jump.
        clr   UP_DEBOUNCER             ; clear debouncers
        inc   DOWN_DEBOUNCER           ; increment db
        cp    DOWN_DEBOUNCER, #22      ; if at 22, then set high.
        jp    ne, SET_UP_AND_DOWN_FLAG ; else, skip.
        ld    DOWN_DEBOUNCER, #255     ; ld DB with 255.
        clr   TAP_CNTR                 ; clear TAP_COUNTER
        jp    SET_UP_AND_DOWN_FLAG

CHECK_FOR_POWER:

        clr   OFF_LFC                  ; reset off counter
        clr   UP_DEBOUNCER             ; clear DB's
        clr   DOWN_DEBOUNCER           ;
        or    P0, #00001000B           ; turn off IR's
        cp    POWER_DEBOUNCER, #03     ; is DB aready zerc?
        jp    uge, CLEAR_LINE_DBS      ; if so, don't write
        clr   POWER_DEBOUNCER          ; clear power debouncer
        jp    TIMER0_RETURN            ;

CLEAR_LINE_DBS:

        clr   POWER_DEBOUNCER          ; clear power debouncer
        ld    STOP_FLAG, #01           ; set stop flag.
        and   TMR,#11111101B           ; disable timer 0

; WRITE TO MEMORY -- TAP_CNTR

        ld    ADDRESS, #03             ; POINT TO ADDRESS THAT CONTAINS
TAP_CNTR
        ld    MTEMPH, TAP_CNTR  ; load temp register with TAP_CNTR byte
        ld    MTEMPL, #00              ; load temp register with 00.
        nop
        call  WRITEMEMORY              ;
        or    TMR,#00000010B           ; enable timer 0
        jp    TIMER0_RETURN

SET_UP_AND_DOWN_FLAG:

        cp    DOWN_DEBOUNCER, #255     ; is DOWN DB high?
        jp    eq, SET_DOWN_FLAG        ; if so, set down flag
        cp    UP_DEBOUNCER, #255       ; is UP DB high?
        jp    ne, TIMER0_RETURN        ; if not, leave interrupt
        ld    UP_AND_DOWN, #01         ; else, set direction for up
        jp    TIMER0_RETURN            ; leave interrupt.

SET_DOWN_FLAG:

        ld    UP_AND_DOWN, #02         ; else, set direction for down

TIMER0_RETURN:

        iret
```